package com.twitter.product\_mixer.core.product.guice

import com.google.inject.Key

import com.google.inject.OutOfScopeException

import com.google.inject.Provider

import com.google.inject.Scope

import com.google.inject.Scopes

import com.twitter.util.Local

import scala.collection.concurrent

import scala.collection.mutable

/\*\*

\* A scala-esque implementation of SimpleScope: https://github.com/google/guice/wiki/CustomScopes#implementing-scope

\*

\* Scopes the execution of a single block of code via `let`

\*/

class SimpleScope extends Scope {

private val values = new Local[concurrent.Map[Key[\_], Any]]()

/\*\*

\* Execute a block with a fresh scope.

\*

\* You can optionally supply a map of initialObjects to 'seed' the new scope.

\*/

def let[T](initialObjects: Map[Key[\_], Any] = Map.empty)(f: => T): T = {

val newMap: concurrent.Map[Key[\_], Any] = concurrent.TrieMap.empty

initialObjects.foreach { case (key, value) => newMap.put(key, value) }

values.let(newMap)(f)

}

override def scope[T](

key: Key[T],

unscoped: Provider[T]

): Provider[T] = () => {

val scopedObjects: mutable.Map[Key[T], Any] = getScopedObjectMap(key)

scopedObjects

.get(key).map(\_.asInstanceOf[T]).getOrElse {

val objectFromUnscoped: T = unscoped.get()

if (Scopes.isCircularProxy(objectFromUnscoped)) {

objectFromUnscoped // Don't remember proxies

} else {

scopedObjects.put(key, objectFromUnscoped)

objectFromUnscoped

}

}

}

def getScopedObjectMap[T](key: Key[T]): concurrent.Map[Key[T], Any] = {

values()

.getOrElse(

throw new OutOfScopeException(s"Cannot access $key outside of a scoping block")

).asInstanceOf[concurrent.Map[Key[T], Any]]

}

}

object SimpleScope {

val SEEDED\_KEY\_PROVIDER: Provider[Nothing] = () =>

throw new IllegalStateException(

"""If you got here then it means that your code asked for scoped object which should have

| been explicitly seeded in this scope by calling SimpleScope.seed(),

| but was not.""".stripMargin)

}